案例 9-多边形边缘填充算法

文档编写：霍波魏

校稿/修订：孔令德

时间 2019~2020

联系方式：QQ997796978

说明：本套案例由孔令德开发，原版本为 Visual C++6.0，配套于孔令德的著作《计算机图形学-基于 MFC 三维图形开发》一书。孔令德计算机工程研究所的学生霍波魏在学习计算机图形学期间，对本套案例进行了升级并编写了学习文档。现在程序的编写和程序的解释都是基于 Windows 10 操作系统，使用 Microsoft visual studio 2017 平台的 MFC（英文版）开发。

一、知识点

1. 边缘填充算法

边缘填充算法通过扫描线与每条边相交求出其交点，再将交点右侧的所有像素颜色全部取为反色。

2. “取补”概念

对于黑白图像，取补就是把白色的像素置为黑色，反之亦然；对于彩色的图像，取补就是将背景色置为填充色，反之亦然。取补的一条基本性质是一个像素求补两次就恢复为原色。

二、案例描述

本案例采用边缘填充算法，利用图像处理中的“取补”概念，对一个多边形进行颜色填充。

三、实现步骤

1. 添加 CRGB 类和 CP2i 类
2. 添加 Cline 类以供绘制多边形和添加栅栏时调用
3. 在 CTestView 类中定义读入点表函数、双缓冲绘图函数、绘制多边形函数和填充多边形函数
4. 在 CTestView 类的 OnDraw 函数中调用双缓冲绘图函数进行绘制

四、主要算法

1.CLine 类

```
public:
    CLine();
    virtual ~CLine();
    void MoveTo(CDC *, CP2i);//移动到指定位置
    void MoveTo(CDC *, int, int);
    void MoveTo(CDC *, int, int, CRGB);
    void LineTo(CDC *, CP2i);//绘制直线(不含终点)
    void LineTo(CDC *, int, int);
    void LineTo(CDC *, int, int, CRGB);
    CRGB Interpolation(int, int, int, CRGB, CRGB);//颜色线性插值函数
public:
    CP2i P0;//起点
    CP2i P1;//终点
void CLine::MoveTo(CDC *pDC, CP2i p0)//移动直线起点函数
{
    P0 = p0;
}
void CLine::MoveTo(CDC *pDC, int x0, int y0)//重载函数
{
    P0 = CP2i(x0, y0, CRGB(0.0, 0.0, 0.0));
}
void CLine::MoveTo(CDC *pDC, int x0, int y0, CRGB c0)//重载函数
{
    P0 = CP2i(x0, y0, c0);
}
void CLine::LineTo(CDC *pDC, CP2i p1)//绘制直线函数
{
    P1 = p1;
    CP2i p;//当前点
    if (P0.x == P1.x)//绘制垂线
    {
        if (P0.y < P1.y)//起点低于终点
            for (p = P0; p.y < P1.y; p.y++)
            {
                p.c = Interpolation(p.y, P0.y, P1.y, P0.c, P1.c);
                pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255, p.c.blue
* 255));
            }
        else
            for (p = P0; p.y > P1.y; p.y--)
            {
                p.c = Interpolation(p.y, P0.y, P1.y, P0.c, P1.c);//颜色线性插值
                pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255, p.c.blue
* 255));
```

```
            }
        }
        else
        {
            double k, d;//k为直线斜率
            k = double(P1.y - P0.y) / (P1.x - P0.x);
            if (k > 1.0)//绘制k>1
            {
                if (P0.y < P1.y)
                {
                    d = 1 - 0.5*k;
                    for (p = P0; p.y < P1.y; p.y++)
                    {
                        p.c = Interpolation(p.y, P0.y, P1.y, P0.c, P1.c);
                        pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                        if (d >= 0)
                        {
                            p.x++;
                            d += 1 - k;
                        }
                        else
                            d += 1;
                    }
                }
                else
                {
                    d = 0.5*k - 1;
                    for (p = P0; p.y > P1.y; p.y--)
                    {
                        p.c = Interpolation(p.y, P0.y, P1.y, P0.c, P1.c);
                        pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                        if (d <= 0)
                        {
                            p.x--;
                            d -= 1 - k;
                        }
                        else
                            d -= 1;
                    }
                }
            }
            if (0.0 <= k && k <= 1.0)//绘制0≤k≤1
```

```
        {
            if (P0.x < P1.x)
            {
                d = 0.5 - k;
                for (p = P0; p.x < P1.x; p.x++)
                {
                    p.c = Interpolation(p.x, P0.x, P1.x, P0.c, P1.c);
                    pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                    if (d < 0)
                    {
                        p.y++;
                        d += 1 - k;
                    }
                    else
                        d -= k;
                }
            }
            else
            {
                d = k - 0.5;
                for (p = P0; p.x > P1.x; p.x--)
                {
                    p.c = Interpolation(p.x, P0.x, P1.x, P0.c, P1.c);
                    pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                    if (d >= 0)
                    {
                        p.y--;
                        d -= 1 - k;
                    }
                    else
                        d += k;
                }
            }
        }
        if (k >= -1.0 && k < 0.0)//绘制-1≤k<0
        {
            if (P0.x < P1.x)
            {
                d = -k - 0.5;
                for (p = P0; p.x < P1.x; p.x++)
                {
                    p.c = Interpolation(p.x, P0.x, P1.x, P0.c, P1.c);
```

```
                    pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                    if (d >= 0)
                    {
                        p.y--;
                        d -= 1 + k;
                    }
                    else
                        d -= k;
                }
            }
            else
            {
                d = k + 0.5;
                for (p = P0; p.x > P1.x; p.x--)
                {
                    p.c = Interpolation(p.x, P0.x, P1.x, P0.c, P1.c);
                    pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                    if (d <= 0)
                    {
                        p.y++;
                        d += 1 + k;
                    }
                    else
                        d += k;
                }
            }
        }
        if (k < -1.0)//绘制k<-1
        {
            if (P0.y > P1.y)
            {
                d = -1 - 0.5*k;
                for (p = P0; p.y > P1.y; p.y--)
                {
                    p.c = Interpolation(p.y, P0.y, P1.y, P0.c, P1.c);
                    pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                    if (d <= 0)
                    {
                        p.x++;
                        d -= 1 + k;
                    }
```

```
                    else
                        d -= 1;
                }
            }
            else
            {
                d = 1 + 0.5*k;
                for (p = P0; p.y < P1.y; p.y++)
                {
                    p.c = Interpolation(p.y, P0.y, P1.y, P0.c, P1.c);
                    pDC->SetPixelV(p.x, p.y, RGB(p.c.red * 255, p.c.green * 255,
p.c.blue * 255));
                    if (d >= 0)
                    {
                        p.x--;
                        d += 1 + k;
                    }
                    else
                        d += 1;
                }
            }
        }
    }
    P0 = p1;
}
void CLine::LineTo(CDC *pDC, int x1, int y1)//重载函数
{
    LineTo(pDC, CP2i(x1, y1, CRGB(0.0, 0.0, 0.0)));
}
void CLine::LineTo(CDC *pDC, int x1, int y1, CRGB c1)//重载函数
{
    LineTo(pDC, CP2i(x1, y1, c1));
}
CRGB CLine::Interpolation(int m, int m0, int m1, CRGB c0, CRGB c1)//颜色线性插值函数
{
    CRGB c;
    c = double(m - m1) / (m0 - m1)*c0 + double(m - m0) / (m1 - m0)*c1;
    return c;
}
```

2.CTestView 类

```
public:
    void ReadPoint();//读入顶点表
    void Graph(CDC* pDC);//双缓冲绘图
    void DrawPolygon(CDC* pDC);//绘制多边形
```

```
        void FillPolygon(CDC* pDC);//填充多边形
protected:
    CP2i P[7];//点表
    int nClientWidth, nClientHeight;//屏幕客户区宽度和高度
    int nHWidth, nHHeight;//屏幕客户区的半宽和半高
    BOOL bFill;//填充标志
    int MaxX, MinX, MaxY, MinY;//包围盒的对角点
void CTestView::ReadPoint()//多边形顶点表
{
    P[0].x = 50;  P[0].y = 100;
    P[1].x = -150; P[1].y = 300;
    P[2].x = -250; P[2].y = 50;
    P[3].x = -150; P[3].y = -250;
    P[4].x = 0;   P[4].y = -50;
    P[5].x = 100; P[5].y = -250;
    P[6].x = 300; P[6].y = 150;
}
void CTestView::Graph(CDC*pDC)//双缓冲绘图
{
    CRect rect;//定义客户区
    GetClientRect(&rect);//获取客户区大小
    nClientWidth = rect.Width();//屏幕客户区宽度
    nClientHeight = rect.Height();//屏幕客户区高度
    nHWidth = nClientWidth / 2;//屏幕客户区半宽
    nHHeight = nClientHeight / 2;//屏幕客户区半高
    if (!bFill)
        DrawPolygon(pDC);
    else
    {
        FillPolygon(pDC);//填充并绘制多边形
        DrawPolygon(pDC);
    }
}
void CTestView::DrawPolygon(CDC* pDC)
{
    for (int i = 0; i < 7; i++)
    {
        if (P[i].x > MaxX)
            MaxX = P[i].x;
        if (P[i].x < MinX)
            MinX = P[i].x;
        if (P[i].y > MaxY)
            MaxY = P[i].y;
        if (P[i].y < MinY)
```

```
            MinY = P[i].y;
    }
    CLine* line = new CLine;
    CP2i t;
    for (int i = 0; i < 7; i++)//绘制多边形
        {
        if (0 == i)
        {
            line->MoveTo(pDC, nHWidth + P[i].x, nHHeight - P[i].y);
            t = P[i];
        }
        else
            line->LineTo(pDC, nHWidth + P[i].x, nHHeight - P[i].y);
    }
    line->LineTo(pDC, nHWidth + t.x, nHHeight - t.y);//闭合多边形
    line->MoveTo(pDC, CP2i(nHWidth + MinX, nHHeight - MinY));//绘制包围盒
    line->LineTo(pDC, CP2i(nHWidth + MinX, nHHeight - MaxY));
    line->LineTo(pDC, CP2i(nHWidth + MaxX, nHHeight - MaxY));
    line->LineTo(pDC, CP2i(nHWidth + MaxX, nHHeight - MinY));
    line->LineTo(pDC, CP2i(nHWidth + MinX, nHHeight - MinY));
    delete line;
}
void CTestView::FillPolygon(CDC* pDC)
{
    COLORREF BackClr = RGB(255, 255, 255); //背景色
    COLORREF FillClr = RGB(0, 0, 255); //填充色
    int yMin, yMax; //边的最小y值与最大y值
    double x; //当前点的X坐标
    int y; //当前点的y坐标
    double k; //k为斜率的倒数
    for (int i = 0; i < 7; i++)//循环访问多边形所有边
    {
        int j = (i + 1) % 7;
        k = double(P[i].x - P[j].x) / (P[i].y - P[j].y); //计算斜率的倒数
        if (P[i].y < P[j].y)
        {
            yMin = P[i].y;
            yMax = P[j].y;
            x = P[i].x; //得到x|ymin
        }
        else
        {
            yMin = P[j].y;
            yMax = P[i].y;
```

```
                x = P[j].x;
            }
            for (y = yMin; y < yMax; y++)//沿每一条边循环扫描线
            {
                for (int m = ROUND(x); m < MaxX; m++)//循环处理每一条扫描线与边的交点
右侧像素
                {
                    if (FillClr == pDC->GetPixel(nHWidth + m, nHHeight - y))//如果是填
充色
                        pDC->SetPixelV(nHWidth + m, nHHeight - y, BackClr);//设置为背
景色
                    else
                        pDC->SetPixelV(nHWidth + m, nHHeight - y, FillClr);//设置为填
充色
                }
                x += k;//计算下一条扫描线的x起点坐标
            }
        }
}
```

五、实现效果

多边形边缘填充算法效果如图 9-1 所示。

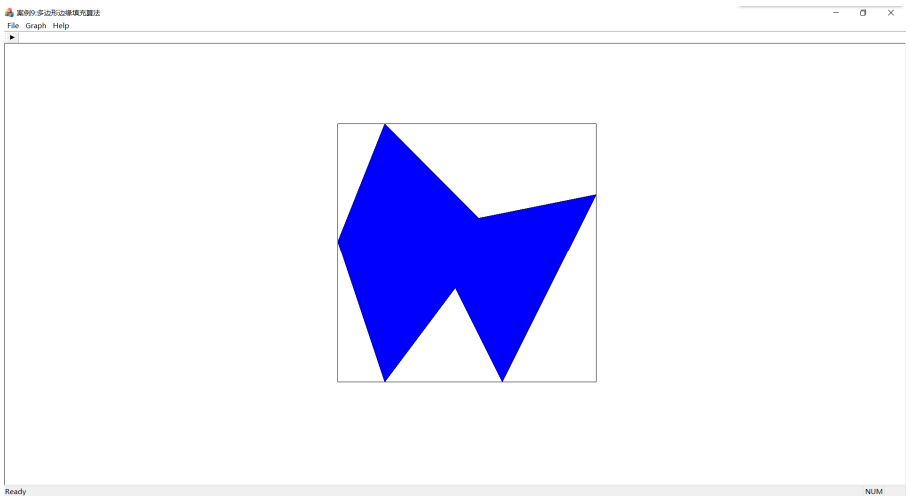

图 9-1 多边形边缘填充算法效果图

六、遇到的问题及解决方案

提高效率的方法有两种：包围盒和栅栏。

包围盒是多边形的外接矩形，可以限制像素取补的范围；栅栏：通常取通过多边形顶点之一的直线，处理每条边与扫描线的交点时，只将交点与栅栏之间的像素取补。

七、案例心得

多边形边缘填充算法可以逐行实现也可以借助于双缓冲技术一次完成。由于该技术不能实现颜色的插值，一般在他乡填充算法中不使用。